SWAPL
SouthWest APL Users' Group

Newsletter

July, 1993

DYALOG APL/W VERSION 6.3

Dyadic Systems, Ltd. has recently announced the 6.3 release of their APL interpreter that runs on 386 and 486 platforms in the Windows environment. The following is an overview of the new features that come with this version.

A new system function □*NA*. This allows calling any DLL (Dynamic Link Library) subroutine from within APL, using APL syntax! A mapping scheme is defined to convert Windows syntax to APL syntax.

Support for color on GUI objects.

Seven new "graphical" objects. These provide the ability to draw: rectangles, circles, ellipses, points, lines, polygons, and filled areas; graphical text; and bit mapped images and icons.

Drag-and-drop has been extended to these graphical objects.

A new Locator object. Allows input or selection in a graphical manner.

Support for icons and bit mapped graphics. These may be used on forms, buttons, and menus (toolbars).

The standard Windows File dialog box.

A new PRINTER object. This allows device-independent printing of text and graphics.

A new CLIPBOARD object. Allows program control of movement of data to and from the Clipboard. A new event notifies the program when the contents of the Clipboard have changed.

Extension of □*NQ*. This will permit sending messages under program control directly to an object in the application.

Improvements to several properties and events. These include: adding a user-definable coordinate system; improving the function of the CHANGED, GOT_FOCUS, LOST_FOCUS, DDE, and ICONIFY events; and allowing scroll events for Forms.

New properties. A new DATA property allows storing any APL array with an object's definition; an INDEX property permits placement of items in List boxes or Combo boxes; an ONTOP property forces objects to the top window position; an AUTOCONF property simplifies resizing events; and a PROPLIST property contains a list of properties associated with an object.

Distributed workspaces. *WDesign* incorporates the new features of this release in the GUI design process; *BMed* allows creating/editing bitmaps; *Graphs* illustrates new graphical objects; and *Arachnid* shows off bitmap and image capabilities.

"Version 6.3 completes the definition of the Dyalog APL GUI support, and will be the basis for the forthcoming OSF/Motif, OS/2 and Windows NT implementations. These versions of Dyalog APL will be 100% compatible with one another and with Dyalog APL/W for MS-Windows. This will permit true portability of APL applications between different GUIs and different hardware platforms."

TABLE OF CONTENTS

Dyalog APL/W Version 6.3	1
SAC '94 (Again)	2
Letters to the Editor	3
Tidbits	4
User Group News	5
Dallas June Meeting Notes	7
APL User Groups (USA)	8
Application Notes [on vacation]	
Working with Manugistics' APLWIN	9
Performance Analysis Summary	16
More on Boolean Flags	19
⍝ Extension Notes	21

SWAPL Officers

President and Program Chair: Margaret Brooker
P.O. Box 210367
Bedford, TX 76095
(817) 577-0165
CompuServe: 73700,2545
Internet: 73700.2545@compuserve.com

Editor and Treasurer: Stuart Yarus
P.O. Box 210367
Bedford, TX 76095
(817) 577-0165
CompuServe: 73700,2545
Internet: 73700.2545@compuserve.com

Librarian: Jon Capps
P.O. Box 551514
Dallas, TX 75355
(214) 270-4252
CompuServe: 73067,3120
Internet: 73067.3120@compuserve.com

Articles

The *SWAPL Newsletter* staff encourages you (and your grandmother from the old country, as long as she still speaks APL) to submit articles for publication. The preferred media is floppy diskette (PC format, 5¼ inch or 3½ inch size). E-mail is quite acceptable. Sneakernet at SWAPL meetings also works very well. The *SWAPL Newsletter* is composed on an IBM PS/2 Model 80 using Ami Professional under Windows 3.1. Articles formatted in Word Perfect, Word for Windows, and WordStar have been easily imported into Ami Pro, as have flat (unformatted) files and data in the Rich Text Format. Printing is done on a Hewlett-Packard LaserJet III.

Advertising

The *SWAPL Newsletter* does take advertising. Our rates are $40.00 for a full page, and are scaled down for multiple insertions and smaller ads. Please contact our acting Advertising Manager (the Editor) for details.

1994 Symposium on Applied Computing (SAC'94)

[The first half of this notice appeared in this newsletter last month. The entire notice is printed this month. (We goofed.) -Ed]

Special Interest Group APL

Call for Papers and Participation

March 6-8, 1994 - Phoenix Civic Plaza

Conference Chair
Hal Berghel — University of Arkansas

Program Co-chairs
Terry Hengl — Knowledge Technology, Inc.
Joseph Urban — Arizona State University

Conference Coordinator
Edmund Deaton — San Diego State University

APL Track Chair
Christopher H. Lee — **Manugistics, Inc.**

SAC Steering Committee

Elias Awad	Univ. of Virginia (SIGBIT)
Hal Berghel	Univ. of Arkansas (SIGAPP)
George Hedrick	Oklahoma State University
Richard Hetherington	Univ. of Missouri-KC
John Hightower	Cal State (SIGCUE)
Abe Kandel	Univ. of South Florida
S. Lakshmivarahan	Univ. of Oklahoma
Irving Montanaz	Brookhaven Labs (SIGFORTH)
William Poucher	Baylor University
Roy Rada	University of Liverpool (SIGBIO)
Lynne Shaw	**Consultant (SIGAPL)**
John Talburt	Univ. of Arkansas-LR
Glenn Thompson	AMOCO
Elizabeth Unger	Kansas State (SIGSMALL/PC)
Joseph Urban	Arizona State University
Roger Wainwright	Univ. of Tulsa
Kam-Fai Wong	ECRC SAC '94

SAC is the annual conference of the ACM Special Interest Groups (SIGs) on Applied Computing (SIGAPP), APL (SIGAPL), Biomedical Computing (SIGBIO), Business Information Technology (SIGBIT), Computer Uses in Education (SIGCUE), FORTH (SIGFORTH) and Small and Personal Computers (SIGSMALL/PC). For the past nine years, SACs have become a primary forum for applied computing practitioners and researchers. Once again, SAC '94 will be held in conjunction with the 1994 ACM Computer

Science Conference in Phoenix. This year, each of the SIGs will have its own track within the conference. Although papers and proposals for panels, tutorials and workshops which deal with the conference themes, STRATEGIC TECHNOLOGY TRANSFER and INTELLIGENT BUSINESS APPLICATIONS, are especially encouraged, papers which deal with all areas of applied computing are solicited. Of particular interest are papers in AI, Biomolecular Computing, Cognitive Science, Computational Biology, Chemistry-Physics and Geosciences, Reliability, Electronic Libraries, MultiMedia, Geographic Information Systems, Image Processing, Logic and Symbolic Programming, Office Automation, Software Reusability and Intelligent Systems. Proposals for special sessions and panels are also encouraged.

IN BRIEF SAC '94 will begin on Sunday, March 6, 1994, with pre-conference tutorials. Paper presentations and panels will begin on Monday, March 7. For further information, contact addresses are listed below.

GUIDELINES FOR SUBMISSION Original papers dealing with any aspect of computing applications will be considered for presentation and publication in the SAC '94 **Proceedings**. This specifically includes these categories of submissions: (1) original and unpublished research, (2) reports of innovative applications in the arts, sciences, engineering, business, government and industry, and (3) reports of successful technology transfer to new problem domains. Each category of submission will be reviewed by peer groups appropriate to that category. Accepted papers in all categories will be published in the **Conference Proceedings**. Expanded versions of selected papers from all categories will be considered for publication in the ACM/SIGAPP quarterly *Applied Computing Review*. In order to facilitate the blind external review process, submission guidelines must be strictly adhered to: Submit 5 copies of original manuscripts to Joseph Urban, Conference CoChair, at the address below. Author name(s) and address(es) **must not** appear in the body of the paper, and self-reference should be in the third person. The body of the paper should not exceed 5,000 words (20 pages, doublespaced) without prior approval. A separate cover sheet should be attached to each copy, containing (1) the title, (2) the author(s) and affiliation(s), and (3) the address (including e-mail and fax) to which correspondence should be sent. Papers relating to SIGAPL, SIGBIO, SIGBIT, SIGCUE, SIGFORTH and SIGSMALL/PC participation should be clearly identified as such so that they may be directed to appropriate track chairs.

All papers and panel proposals must be received by September 1st, 1993. Author Notification will be made by November 1st, 1993. Camera-ready copy must be provided by November 20th, 1993. Send submissions and proposals for panels, tutorials and workshops to one of the Program Co-Chairs: Joseph Urban, Dept. of Computer Science and Eng., Arizona State University, Tempe, AZ 85287 (voice: 602 965 2774; fax: 602 965 2751; jurban@asuvax.eas.asu.edu) Terry Hengl, *PC AI Magazine*, 3310 West Bell Road, Suite 119, Phoenix, AZ 85023 (voice: 602 971 1869;fax: 602 971 2321; 0004077516@mcimail.com)

For more information regarding the APL track, contact the APL Track Chair: Christopher H. Lee, Manugistics, Inc., 2115 E. Jefferson Street, Rockville,MD 20852-4999 (voice: 301 984 5117; fax: 301 984 5094; 71542.606@compuserve.com)

LETTERS TO THE EDITOR

Dear Editor:

Thank you for another portion of excellent APL "food". I mean the May issue of *SWAPL Newsletter*, of course. As usual, I kept it closed until midnight, to relax with after work. Well, I opened it just in time and ... I hope, Margaret Brooker (and copyright law) will forgive me for using the following two lines from her "The Night Before Implementation" poem: "And what to my wondering eyes should appear / But a super program by Gregg to forget my beer". Yes, I am delighted with the connect-the-dots line drawing problem. First of all I liked the problem itself as a simple, but clear and clean model to demonstrate APL power. I jumped to page 15 to see the solution. Yes, the table of all possible decisions is a good and disciplined approach to solve problems. It is used in Nuclear Power Plant Diagnostics, for example. The main thing one can see from Table 1 is that the index of appropriate drawing character is equal to the decimal representation of four bits of features (right, left, below, above). Unfortunately this is not used in `CTDOTS` function and I decided to put an extension to Gregg Taylor's "house of extension notes". As 99% of the work had been already done by Gregg, I named my variant of `CTDOTS` function as `GREGG`. Here it is. (`⎕IO ← 0`)

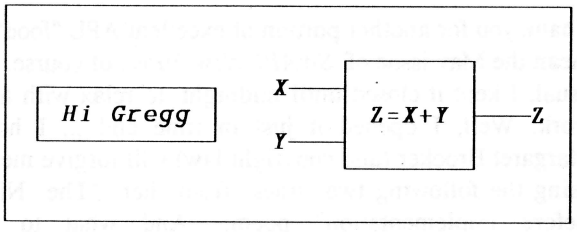
```
[0] Z←A GREGG B;D
[1] ⍎(0=⎕NC 'A')/'A←''*'''
```

```
[2] D←⎕AV[180 180 180 180 197 218
192 181]
[3] D←D,⎕AV[197 193 219 196 197
194 195 198]
[4] B←(-2+ρB)↑(1+ρB)↑B←(Z←B)∊A,D
[5] B←2⊥⊃(1⌽B)(¯1⌽B)(1⊖B)(¯1⊖B)
[6] B←1 1↓ ¯1 ¯1↓B
[7] ((,Z∊A)/,Z)←D[1+(,Z∊A)/,B]
```

The main difference is in line 5, where after rotating the matrix right, left, below and above and disclose, the 2 decode primitive function is applied to get indices of the drawing characters. And of course I use atomic vector indices of pseudo-graphic characters for APL2/PC. For example, try something like the following:

```
      T
************************************
*                                  *
*                  ---------       *
* ************   X---I       I     *
* * Hi Gregg *       I Z=X+Y I---Z *
* ************   Y---I       I     *
*                  ---------       *
*                                  *
************************************
```

```
      '*I-' GREGG T
```



Some final comments. Of course, it is not a talk on graphics at all. I agree with Gregg absolutely that "this approach to problems has applicability beyond this particular function". I only would like to underscore that there is one more thing besides a table of all possibilities. It is a space dependency! Look at a toy problem of writing instructions for a robot. Let us show the map of a road like

```
      M
0 1 1 1 1 1 0 0
0 0 0 0 0 1 0 0
0 1 1 1 0 1 0 0
0 1 0 1 1 1 0 0
0 1 0 0 0 0 0 0
0 1 1 1 0 0 0 0
```

The instructions can be go forward, turn left, turn right, stop. What do you think of a function like

```
      1 2 GO M
FFFFR
    F
LFL F
F RFR
F
  LFS
```

where the left argument is the coordinates of the starting point?

Please, extend my "thank you" to Gregg Taylor for his very nice example demonstrating the power of APL. Thanks from my students, also.

Regards,

Alexander Skomorokhov
Obninsk, Russia

Even an APL guru needs a vacation. Especially a working APL guru. But if you want it to be less expensive and more enjoyable, call Mike von Tiesenhausen as early in the game as possible to plan it or just to dream about it.

Sundance Travel (214) 352-4650

Personalized Service for APL gurus

TIDBITS

Donald McIntyre recently received an honorary Doctor of Science degree from Pomona College. Dr. McIntyre, who earned B.Sc., Ph.D., and D.Sc. degrees from Edinburgh University, taught geology for 35 years at Pomona. He is now "retired" and lives in Scotland, where he works on J.

Manugistics has announced their annual APL*PLUS Users' Conference, to be held September 20 and 21 at the Marriott Hotel in Bethesda, Maryland. Further information will be forthcoming.

Curtis Jones, who lives in San Jose, California, recently returned from a trip to Australia. He sent us a photograph of himself reading a copy of the *SWAPL Newsletter* in Coonabarabran, New South Wales, to show that our newsletter does indeed get around! Thank you, Curtis -- and thank you for the copy of the cover of the magazine *APL Today - Australian Presbyterian Living.*

User Group News

[This information comes from a variety of sources, mostly newsletters and attendee reports. -Ed]

Atlanta (SEAPL) Sunday, July 25, 1993, is the date for SEAPL's Summer picnic. It will be held from 2 PM to 5 PM at Stone Mountain Park. Details to follow in the next SEAPL newsletter.

Dallas (SWAPL) The June meeting featured Gregg Taylor, who presented the current version of Dyadic Systems' new release of Dyalog APL /Windows, version 6.3. See the review beginning on page 6.

Also at the June meeting, we voted to send $150.00 to the APL93 Russian Fund.

Jon Capps announced that he has an APL bulletin board just about ready to go public. The telephone number is (214) 682-9656. Access is at 8-N-1, up to 2400 baud, for 24 hours a day. There will be no charge to access the BBS. Current SWAPL members will have download and upload capabilities. Although there is currently nothing on line, that will change shortly; the software libraries from the last several annual APL conferences will be loaded, as well as other materials. There is also a forum for messages. The BBS software is being changed as this note is being written (June 23), so the BBS may not be up at various times over the next few weeks as the new program is tuned. Stay tuned for further developments.

The July meeting will be held on Tuesday the 6th, at 7:30 PM, at the DeVRY Institute of Technology, 4250 North Beltline Road, Irving, Texas, in Room 218. From Highway 114, go two miles south on Beltline. The building is located on the east side of the road.

The July program features Todd Marshall, who will demonstrate the new release of Manugistics APL*PLUS II/386, Version 5.1. Todd will concentrate on the Windows capabilities of the package (both at the APLGUI and API levels). He will demonstrate the features through an application interfacing with the CIA World Book Database (2.5 megabytes) of 78 categories of information on 268 countries.

A pre-meeting dinner will be at 6 PM at the Black-Eyed Pea restaurant on Beltline Road at Northgate Road, about one mile south of DeVRY. Everyone is invited!

New York (NY/SIGAPL) The June 23 meeting featured Adam Kertesz talking on "Distributing APL Computation".

Northern California (APL BUG) The June 14 meeting featured Arthur Whitney, recently of Morgan Stanley, talking about "A". Meetings are at 7 PM at the Syntex Laboratories building A2, Room 1, 3401 Hillview Avenue, Palo Alto.

David Liebtag, of IBM Corporation, will talk about APL2 for OS/2 on July 12.

Southern California (SOCAL-SIGAPL) The June 22 meeting featured Dr. Jim Brown of IBM Corporation. Jim talked on what IBM has been doing with APL2 for a variety of systems and platforms.

Toronto Come to APL93, August 15 - 19! The next regular meeting will be on September 27, 1993, and will include a review of APL93.

Meetings are on the fourth Monday of each month from September through May, excluding December, and are held at CIBC's 10th floor lounge in the BCE Place building, 161 Bay Street.

Washington, DC (Potomac ACM SIGAPL Chapter) On Monday, June 21, 1993, at 7:30 PM, the Potomac ACM SIGAPL, jointly with the APL Special Interest Group of the Capital PC User Group, presented a talk from the ACM Lectureship Series. Bob Bernecky talked on Ergonomics and Language Design. Application of ergonomic, or human engineering, principles to computer language design improves the effectiveness of computing. Traditional programming languages, such as FORTRAN and C, are semantically deficient in that their von Neumann pedigree fails to reflect or enhance people's patterns of thought. This ergonomic gap between thought and programming language is detrimental to the effective use of computers. In contrast, semantically advanced languages enhance our thought processes and hide superfluous detail, allowing even mediocre programmers to make effective use of computers. Furthermore, such languages simplify the

-- NOTICE --

Kathy Parker (formerly with Decision Consultants, Inc.),

announces the opening of new offices:

Mail: Kathy Parker & Associates
14232 Marsh Lane, Suite # 343
Dallas, Texas 75234

Office: 214-386-0293 (voice)
214-386-0292 (fax)

(Voice mail messages and faxes are also retrieved after regular business hours and on weekends.)

Specializing in contract and permanent positions in:

APL/APL2	SAP
C, with DOS, OS/2, or Unix	Unisys 1100/2200
Oracle	IEF
C++	AS/400
PowerBuilder	MS and X Windows
Artificial Intelligence	Open Systems
Sybase	Informix
LAN/WAN/MAN	Banking/Financial
Documentation	Information Development
MSA / D & B	Hogan
Project Management	Foundation CASE
Cobol, CICS	OSF/DCE & DME
Client/Server	SNA
CICS Systems	LU 6.2

Positions open in many major cities for candidates with a minimum of one year's industry experience.

Call for current and future openings; mail or fax resume in strictest confidence.

compiler writer's job, increase the portability of programs and programming skills across computer architectures, provide an excellent base for programming parallel computers, improve performance, and increase the probability of writing fault-free programs. Examples of the power of such languages were taken from the literature, and expressed in the J dialect of APL.

Robert Bernecky is a language designer and implementor of APL systems. Formerly Director of Research at I.P. Sharp Associates, he left there in 1990 to found Snake Island Research. Inc. His current activities include the creation of high-performance compilers for parallel computers. Robert Bernecky is an ACM Lecturer for 1992-93.

Future Meetings: Monday, July 19, 1993 at 7:30 PM: "APL Resources and the Internet". Monday, August 16, 1993 at 7:30 PM: Nancy Wheeler on APL2. Monday, September 20, 1993 at 7:30 PM: "Highlights of APL93". Monday, October 18, 1993 at 7:30 PM: Peter Lewis on AGSS.

APL SIG meetings are held at the Capital PC User Group headquarters at 51 Monroe St., Lobby Level, Plaza East 2, Rockville, MD 20850. To find the meeting and for late information: contact PotAPL@acm.org. Late announcements and directions are also available by calling Capitol PC Users Group (CPCUG) Voice Information at (301) 762-6775.

Dallas June Meeting Notes

Todd B. Marshall
Conroe, Texas

It has become a Windows World and APL must not be shut out of it. The end users tell us (by how they spend their software dollars) that they would rather point and click. As anyone who has tried writing a windows application can quickly confirm, creating a "true" Windows application is not a trivial task. The more easy software is for the users to operate, the more skillfully it has been crafted by the developer.

This month we got a look at Dyadic Systems' Dyalog APL, Version 6.3, with Windows enhancements. Gregg Taylor gave us our first look at Dyalog's magic...and it truly looks magical.

Gregg performed his wonders using a 50 megahertz 486 PC, and the responsiveness was impressive. He started with a clean form and Dyalog's tool box of 30 icons. This tool box allows you to specify a type of object and sculpt it onto the form. You place it, you shape it, and then you embellish it with properties and make it responsive to events. Most of this is done simply by pointing, dragging, selecting, and clicking. In a very few instances, Gregg even used the keyboard.

The form, once created, can be saved in one of two forms. It can be saved in the Workspace or it can be saved as a Function. If you save it in a workspace, the Dyalog smoke and mirrors hides it away. If you save it in a function, it creates a function that when executed, produces the form. You can edit this function and change the behavior and appearance of the form. In this respect it seems to behave a little like Visual C++.

As with most object oriented programming requirements, Dyalog has adopted a hierarchical naming style. These compound names are built up with a period (".") separator between the levels. For example, you may have code like `'FORM.BUTTON' ⎕WG 'SIZE'` to reference the size property of a button on a form.

At this point I must suppress the temptation to turn this into a product review and go back to Gregg's demonstration. To give his treatment some cohesion, Gregg selected a graphing application. His form would allow selection of a graph style (e.g. line, pie, or bar) through a combo-box. It would accept data through a two column array of edit boxes. It would even accept a formula in an edit box; sundry buttons were also available. It would then display the results in a sub-window. If your request gave it indigestion, it would throw up a "stop" dialog box with a helpful message.

Dyalog's strategy seems to be to create function shells for event processing. After setting up the form and specifying the events to be handled, the APL programmer edits the resulting function shell providing the application functionality. After demonstrating this concept (it only took about 10 minutes), Gregg showed us his completed product.

By browsing through the code, it was evident that Dyalog has distilled the APL interaction with Windows through 6 or 7 system functions. For example, there is a `⎕WC` to create an object. There is a `⎕WS` to set a property. There is a `⎕WG` to get a property. There is a `⎕DQ` (read de-queue) which among other things seems to be the implementation of the Windows Wait Loop. These system functions are dyadic and return results. Their arguments are supplied by nested arrays. Parameters may be "named" or "positional" as the user chooses.

Gregg demonstrated the power of the Dyalog environment by tracing his code line by line. He could display functions and variables just by clicking on their names in the code. The environment was completely Windows. Each displayed function appeared in a new window. Each displayed variable appeared in a window. These windows, like any other window could be moved and sized at will. The focus moved as desired with a simple click on the window of interest.

We asked Gregg to tell us how he felt about Dyalog APL as compared to Visual Basic. He said that he liked setting up the forms in Visual Basic but liked working with the logic in Dyalog APL. While he wouldn't admit it, this seemed to some observers to show a preference for APL over Basic (blind or visual).

Unfortunately, you can't have your cake and eat it, too. Dyalog APL seems to stand alone. It only has the Windows functionality that Dyadic implementors choose to enable. [*□NA* allows you access to all Windows DLLs. See overview on page 1 of this newsletter. -Ed] As Windows changes and Dyalog APL matures, you will get the new functionality from the factory...not by making extensions yourself...and therein lies a problem in Gregg's opinion.

Gregg says that Dyalog APL will always be about a year behind Windows. Further, he noted that there are all kinds of "add-on" products you can purchase for Visual Basic (the product even comes with a catalog of them). This can't happen with Dyalog APL because their architecture is not open. With this closed strategy, the implementers limit the number of people that can get around the problems. They limit the creativity and effort that can be applied to its extension. Gregg sees this to Dyalog's detriment.

Thanks to Gregg for an excellent presentation. As some of us can't afford the $1,700 to try each company's APL offerings, such demos are a great service SWAPL meetings provide attendees.

APL User Groups (United States)

The following is a listing of all known cities in the United States with active APL user groups, along with the name of a person to contact and at least one way to contact him/her. Not all user groups have newsletters, but all do meet periodically. If you know of any additional groups (that would like to be listed), please let the editor know at his masthead address. Updates to contact information are most welcome.

Atlanta, Georgia
Doc Manges
991 Creekdale Drive
Clarkston, GA 30021
(404) 292-4968

Chicago, Illinois
Larry Mysz
836 Highland Drive
Chicago Heights, IL 60411

Dallas, Texas
Margaret Brooker
P. O. Box 210367
Bedford, Texas 76095
(817) 577-0165
CompuServe: 73700,2545
Internet: 73700.2545@compuserve.com

Hartford, Connecticut
Bob Pomeroy
Mass Mutual Life
1295 State Street
Mail Drop F465
Springfield, MA 01111
(413) 788-8411, ext. 2838

New York, New York
Jacob Brickman
P. O. Box 138
New York, NY 10185-0002
(718) 773-4093

Northern California
Curtis A. Jones
228 South 15th Street
San Jose, CA 95112-2150
Internet: jonesca@vnet.ibm.com
Prodigy: CJGH56A
FidoNet: 1:143/11

Southern California
Roy A. Sykes, Jr.
Sykes Systems, Inc.
4649 Willens Avenue
Woodland Hills, CA 91364-3812
(818) 222-2759

Washington, District of Columbia
John A. Martin
(301) 497-2698
fax: (301) 498-8260
Internet: jmartin@milo.starlab.csc.com

WORKING WITH MANUGISTICS' APLWIN

Todd B. Marshall
Conroe, Texas

[This is the second part of a two-part article. The first part appeared in the June issue of this newsletter. The code referenced is also in the June issue. The window Todd creates is on page 14. -Ed]

How the program works

From here on, I will be referencing lines of code in the function *HELLOWIN*. The dialog begins with APL invoking the *W_Init* function and capturing the result (Line 14). This is analogous to a "C" program named "WinMain" being called by Windows . The agent is the WinMain program and has actually already been called by Windows. *W_Init* just sends APL the arguments.

Lines 17-20 create what Manugistics calls a filter. The function *W_FP* is a little nested array builder. It is capable of building a hierarchical nested array suitable as an argument to the *W_CreateFilter* function (line 20). Line 17 (a monadic call to *W_FP*) is kind of the header for the filter. It tells the Agent in what section of the APLWDEF.INI file to look for message types (the WP means look in the [Pattern WP] section of the file). Line 18 says if the agent gets a WM_PAINT (left argument) message, execute the string (right argument).

In this example, the string is formed by the program *THISFN*. WM_PAINT is an argument to *THISFN*. In the context of our APL program, WM_PAINT is a line label (line 91) and therefore is just the integer 91. *THISFN* takes this and generates the string "HELLOWIN 91". It gets the *HELLOWIN* (the name of "this function") by looking to the state indicator.

If the agent receives a WM_PAINT message, it consults the filter we are defining and sees that it should execute the "HELLOWIN 91" string. This causes a recursive call to *HELLOWIN*. The *HELLOWIN MSG* argument on entry is now 91 and this is interrogated on line 4. Seeing that *MSG* is not 0, line 4 says jump to *MSG* (jump to line 91). Low and behold, line 91 is the entry point to our code for dealing with the WM_PAINT message.

You don't have to use this recursive technique I illustrate here; Manugistics doesn't. But I certainly prefer it to having little functions strung out all over the place to handle these messages. In one compact program, I have implemented the *HELLOWIN* program. Manugistics' examples use two or three functions.

We kind of ran off with ourselves trying to describe filters, but you now see the whole chain of events.

Line 19 defines filtering for the *WM_DESTROY* message. Line 20 gives the filter definition to the agent by calling *W_CreateFilter* using "w_fp" as an argument. "w_fp" is the global created by the *W_FP* function. There will be a little delay here as the agent "compiles" the filter information. It probably builds a table in memory with message numbers and a list of pointers to strings to be executed. But the message could be coming from any number of windows. We need a table and a filter for each window. In this example we only have one window, so no problem.

W_CreateFilter returns a reference to the filter we defined (line 20) and we save this reference in the variable WndProc (the same name Petzold's example uses for his message handling program ... hmmm).

When we create the window to use this filter, we supply WndProc as if it was a pointer to a "C" function. Here's some more of Manugistics' magic. The Agent gets a WM_PAINT (15) message and Windows says it's for WndProc. The agent goes to the WndProc table, finds the WM_PAINT (15) reference, and tells the APL program to execute "HELLOWIN 91". This execute actually happens on line 77 within the *W_Wait* function.

Registering the window class

In "C", you build up a structure called a WNDCLASS to collect information for creating a window. We do the same thing with APLWIN, building a nested array instead of a structure. Lines 23-50 illustrate how this nested array is formed. I have shown the corresponding "C" code to illustrate how closely it correlates to APLWIN. Manugistics supplies a function *W_A* to help you build up nested arrays (creates a global w_a).

The definitions for these structures and parameter descriptions can be found in **Microsoft Programmer's Reference Library Volume 3: Messages, Structures, and Macros**. If you're using Borland's C++, it is in **Windows API Reference Guide, Volume 2.**. You need to have one of these or a similar reference to know what's going on in Windows.

Lines 24, 39, 42, and 45 divulge a little of the subtle elegance of the implementation of the Agent.

On line 24, we are essentially setting switches (turning on bits in a 16 bit word). In "C", CS_VREDRAW and CS_HREDRAW are #defined as 0x0001 and 0x0002... bits 1 and 2). By "or"ing these we set the switches in the

word. But APL doesn't have a "bitwise-or". Manugistics has enabled us to reference these mnemonics as chararacter strings. The agent determines the bits of interest and does the bitwise-or for us...elegantly simple.

On line 39, we reference a function `Load_IDI`. There is no such function in Windows. Windows wants to see:

(LoadIcon 0 IDI_APPLICATION)

In APLWIN we could write:

(W_Call 'LoadIcon' 0 (W_C'IDI_APPLICATION'))

Two undesirable things are happening here. Firstly, we are calling the function `W_C` with the argument 'IDI_APPLICATION'. Secondly, we are waking up the agent with our `W_Call` to LoadIcon. We would rather just send some data to the agent in this position in the structure and call him once for the whole structure.

`W_C` is a service function provided in APLWIN to look up constants. If we give it a name, it returns a number. If we give it a number, it returns name(s) #defined to that number. If we give it a string of names, it returns the bitwise-or of the integer values referenced by the names in the string. If we say " `'WM' W_C 1 2 3` ", it returns a matrix WM_CREATE; WM_DESTROY; WM_MOVE because they begin with 'WM' and the values for these messages are 1, 2, and 3 respectively. It's a neat function but takes up 32K of workspace. And looking all this up in its tables takes time. Manugistics has provided a more efficient way of accomplishing the same thing for production code.

Alias Functions

Manugistics provides a compromise solution in what it calls ALIAS functions. These are non-Windows functions defined in APLWDEF.INI which translate into valid windows functions. It's like using macros. In this case the macro becomes (Load_IDI 'IDI_APPLICATION')

The Petzold "C" code for line 39 is:

LoadIcon (NULL, IDI_APPLICATION)

But WINDOWS.H #defines IDI_APPLICATION as MAKEINTRESOURCE(32512). The header file goes through three or four more macro expansions. Finally, the zero terminated character string pointer that LoadIcon is looking for as its second argument is not a pointer to a character string at all. It is a double word (0,32512). What we really have is a "C" call for this special reference to a system icon becoming:

LoadIcon (0,0,32512)

So we could have written line 39 as ('LoadIcon' 0 32512) and everything would have been fine. But it's not understandable. The day after you write it, you ask, "what is 32512?" To fake the system out and still keep things mnemonic, Manugistics introduces alias functions.

"Load_IDI" is Manugistics' alias for "LoadIcon (0,~W, W resource)" where "resource" is the mnemonic for the system icon resource (i.e. IDI_APPLICATION, IDI_ASTERISK, etc.) If you look in the APLWDEF.INI file you will find the line:

Load_IDI=H(~H, ~W, W idc_icon) ALIAS LoadIcon

So our Load_IDI reference becomes a LoadIcon of the form:

LoadIcon=H(~H,~W,W idc_icon)

Looking in APLWDEF.INI we see that IDI_APPLICATION is defined as 32512, so ultimately our call becomes:

'LoadIcon' 0 0 32512

As you can see, to really understand this stuff you must first understand what is happening in the "C" program. You can then easily make guesses about how the Agent is handling the situation. Sometimes your guesses are even right.

Line 45 illustrates another shortcut. This line could have read:

W_Call 'GetStockObject' 'WHITE_BRUSH'

which would place the handle result from GetStockObject in this position of the nested array (wndclass structure). But here again, this would cause us to make a special imposition on the agent and that takes time. However, if the agent sees a nested array in the structure element, it quietly does the `W_Call` on that nested array element itself. So anything you put here that you could use W_Call on directly (don't read this `⍎'W_Call',` argument) then you've met the requirements.

Line 50 alludes to one final subtle point. Recall that I said APLWIN doesn't support multiple instances of your application. Each instance must have a different name.

So Manugistics recommends that you form the name of the class you are registering with an expression like:

```
szAppName←'HelloWin',⍕hInstance
```

If you look at my line 6, I create names of the form:

```
szAppName←'HelloWin',⍕?1000
```

I do this because I found that while experimenting and debugging, I could get a class registered but not be able send the WM_QUIT message to get rid of it. When this happens, I must change my program to use another name to avoid a "WIN REGISTRATION ERROR." Having some automatic randomness in the name alleviates this annoyance. After debugging and experimenting are concluded, I recommend you use Manugistics' practice.

Finally we are to line 53. In a "C" program we would call RegisterClass with a pointer (&wndclass) to the structure we just created. We can't do this in APL, so APLWIN gives us a function *W_RegisterClass* to do the same thing. The expression "⍎ *W_A* ⍎" causes the global "w_a" to give up its contents and clears it to an empty vector. I assign it to a variable (wndclass) here for debugging reasons. If *W_RegisterClass* discovers a problem in its argument, the argument may no longer be easily available for inspection (the argument name is "wc" but I usually forget). In production code you would make no assignment here.

Creating a window

Well. Now Windows knows we exist, and we can start to make use of its services. The first service we need is putting a window on our screen. The CreateWindow procedure is one of Windows more heavy duty functions. It wants a structured argument just like RegisterClass did. But this time we can call it with the *W_Call* function because no special pointers are needed. Line 56 begins setting up the call with the name of the Windows procedure CreateWindow. The nested array is then built up with the arguments to *CreateWindow* using the convenient *W_A* function.

Our ticket to CreateWindows services is the application name we just registered (szAppName). It is the first item in the nested array structure we need (line 57). Then we have the caption on the window (line 58).

Line 59 defines the window's style. In our example we are putting up a simple overlapping window. *WS_OVERLAPPEDWINDOW* gives us *WS_OVERLAPPED*, *WS_CAPTION*, *WS_SYSMENU*, *WS_THICKFRAME*, *WS_MINIMIZEBOX*, and *WS_MAXIMIZEBOX* with one simple reference. You could have defined these styles individually by giving them all in a single character string separated by blanks.

It is here that we may start to see the value of working in APL to learn the Windows API. You might reference an APL variable on line 59 instead of the literal 'WS_OVERLAPPEDWINDOW' (say you call the variable "style"). You could then experiment by assigning different style strings to "style" and run *HELLOWIN 0*. You could run several experiments before a "C" programmer could finish compiling and building his first trial. It is in doing these sorts of things that you redeem the little extra effort expended to learn APL.

CreateWindows returns a handle to the window (hwnd). We need this for reference in any functions affecting our window.

Displaying the window and saying Hello!

Things get exciting now. Line 71 puts the window onto our screen (*ShowWindow*). Line 74 (*UpdateWindow*) causes a WM_PAINT message to be sent to the window. The filter we set up on line 18 says that the agent should execute *HELLOWIN 91*. This will be a recursive call (*HELLOWIN* called within *HELLOWIN*) taking us to the line labeled WM_PAINT (line 91) and we begin painting the window.

If we look at the State Indicator at this point, it shows:

```
HELLOWIN[91]*
⍋
W_CallEach[14]
W_Call[3]
HELLOWIN[74]
```

Painting the window

Our paint messages come from several different sources. First, when we call *UpdateWindow* (line 74) we are asking that a WM_PAINT message be sent. When we register the window with the REDRAW switches set (line 24), we will get a paint message when our window is resized. We don't get a paint message if our window is simply moved.

As noted earlier in the explanation of message cracking, we expect w_args to contain a handle to our window at this point. We capture that handle on line 92 to remain consistent with the Petzold example. On line 94 we prepare the window for painting with a *W_Call* to

BeginPaint. Notice that the second argument is an empty character vector.

Looking in the APLWDEF.INI file for the BeginPaint command syntax we see:

```
BeginPaint=H(HW,>PAINTSTRUCT)
```

This tells us that the BeginPaint procedure will return a handle (H). As arguments, it expects a handle word (HW) and a pointer (>) to paint structure (PAINTSTRUCT). The " > " pointer symbol means that the agent will also return the PAINTSTRUCT structure as a character vector. PAINTSTRUCT is a reference to a data type specification (just like HW is a data type specification reference). Again looking in APLWDEF.INI, we find PAINTSTRUCT in the [type] section defined as:

```
PAINTSTRUCT={H,B,RECT,B,B,{Y,Y,Y,Y,Y,Y,Y,Y,Y,
Y,Y,Y,Y,Y,Y,Y}}
```

Consulting Windows references for the PAINTSTRUCT structure (or looking in WINDOWS.H) we find the following layout:

```
typedef struct tagPAINTSTRUCT { /* ps */
    HDC hdc;                    /* H */
    BOOL fErase;                /* B */
    RECT rcPaint                /* RECT */
    BOOL fRestore;              /* B */
    BOOL fIncUpdate;            /* B */
    BYTE rgbReserved[16];  /*
Y,Y,Y,Y,Y,Y,Y,Y,Y,Y,Y,Y,Y,Y,Y,Y */
    } PAINTSTRUCT
```

The (/*...*/) sequences "C" comments.
I have shown the APLWIN type specifications against the "C" structure specifications so you can see how they directly relate. RECT is another data type specification with the following layout:

```
RECT = {I,I,I,I}
```

The corresponding "C" structure is:

```
typedef struct tagRECT { /* rc */
    int left;          /* I */
    int top;           /* I */
    int right;         /* I */
    int bottom;        /* I */
}RECT;
```

The result of the *W_Call* on BeginPaint is a two item nested array. The first item is the handle to the device context. The second item is the PAINTSTRUCT structure (as a character string). We use strand notation to separate the items into the variables we will use:

```
  [93]  A hdc      =           BeginPaint
(hwnd, &ps);
  [94]          (hdc   ps)    ←   W_Call
('BeginPaint' hwnd   '')
```

A *W_Call* on GetClientRect, tells us the size of our window. We need this to know where to paint. The layouts can be researched in the same fashion used above:

```
  [95]   A
GetClientRect (hwnd, &rect) ;
  [96]          (rect)    ←   W_Call
('GetClientRect' hwnd   ''  )
```

"rect" now is a character vector 8 bytes long. As we have seen, it is made up of four 2-byte integers. If we wanted to see the integer values, we could use the Manugistics supplied function *C2I* which converts characters in pairs to integers:

```
irect ← C2I rect
```

A complementary function goes the other way:

```
rect ← I2C irect
```

For our needs here we are satisfied to leave "rect" in its character form for we will only use it as an argument to the DrawText procedure. DrawText is a powerful Windows function for throwing things into a rectangular area on the screen:

```
  [98]   A       DrawText  (hdc,'Hello,
Windows!",-1,&rect,
  [99]   A                 DT_SINGLELINE |
DT_Center | DT_VCENTER) ;
  [100]  W_A  'DrawText'  hdc  'Hello,
Windows!' ¯1 rect
  [101] W_A          'DT_SINGLELINE
DT_CENTER   DT_VCENTER'
  [102] t←W_Call ⍬ W_A ⍬
```

We're using it in a trivial way. The first argument "hdc" is the device context we got from BeginPaint. Then we have a character string which we wish to draw in the rectangle. Many of the Windows procedures require us to tell how long the string is. The -1 means the string is terminated by a 0. When you see the 0, you're at the end of the string. "rect" is given just as we received it from GetClientRect. If we look at the DrawText syntax in the APLWDEF.INI file we see the following:

```
DrawText=I(H,*C,I,>RECT,U)
```

This tells us an integer (I) is returned. It is the height in pixels of the text. The first argument is a handle (H-hdc). The second argument is a pointer(*) to a character (C) type. The third argument is an integer(I-length of the string pointed to in the previous argument). Then comes our pointer to our "rect" (>RECT) structure. Again, the (>) tells us this structure will be returned. In our previous visits to these structure pointers, we used empty character vectors (' ') in this position. This told the agent to create the structure according to type specifications and return it. In this case, we have the structure but it will also be returned by our call to DrawText. In "C", arguments passed by pointer can be referenced and written to directly. In APL, we can't do that so we return a copy of the new version. The last argument (U) is an unsigned integer word. It contains switches which control how the text will be drawn. Here we ask to have it on a single line (DT_SINGLELINE) and centered (DT_CENTER) on that line. We want the single line vertically centered (DT_VCENTER) as well. We have no use for the result so we just trap it in "t".

On line 104, we say we have finished painting in the window by calling EndPaint. Notice that this call requires the PAINTSTRUCT structure "ps" that we captured on our call to BeginPaint. For every BeginPaint there must be and EndPaint. Presumably painting can be nested and this pairing makes such nesting clear.

After our window is initially painted by `UpdateWindow` (line 74), we get subsequent paint messages from our wait loop (line 77). In these cases our state indicator is:

```
HELLOWIN[91]
⍎
W_Wait[16]
HELLOWIN[77]
```

So it is on line 16 of `W_Wait` that the string sent by our WM_PAINT filter statement actually get executed.

Returning after handling messages

In "C", a return is made to windows using the "return" statement (line 107). If the application processes a message, the message dictates what value should be returned from the procedure. WM_PAINT (as with most messages) requires 0 zero to be returned. APL generally returns from its function by setting the result variable given on the header line. Well, in Windows you don't do it that way. Rather, APLWIN uses the global "w_z" for the return value. On line 108, we set this variable to 0 as instructed and Exit our function. The default value for w_z is 0, so it is not necessary that we make this assignment.

Destroying the window and exiting

When you select Close (or use the ALT-F4 accelerator), a WM_DESTROY message is sent through the filter. Our program requests that the string 'HELLOWIN 112' be executed in this instance. This brings us to the WM_DESTROY label which is the entry to our window destroying process. You must filter for this message. Remember, `W_Wait` stays in the loop until it sees a WM_QUIT message. This message comes from the PostQuitMessage procedure (line 114). If you have no way of getting the WM_QUIT message to the wait loop, your program will appear to hang.

To recover, you can usually use ALT-TAB to get to the Agent Window. Clicking on the Debug APL item in the System Menu will send an interrupt to the W_Wait function. This will create a WIN BREAK error event and exit `W_Wait`. ALT-TAB to the window and close it if it's still there. ALT-TAB to the APL session and you should see your program suspended.

Debugging

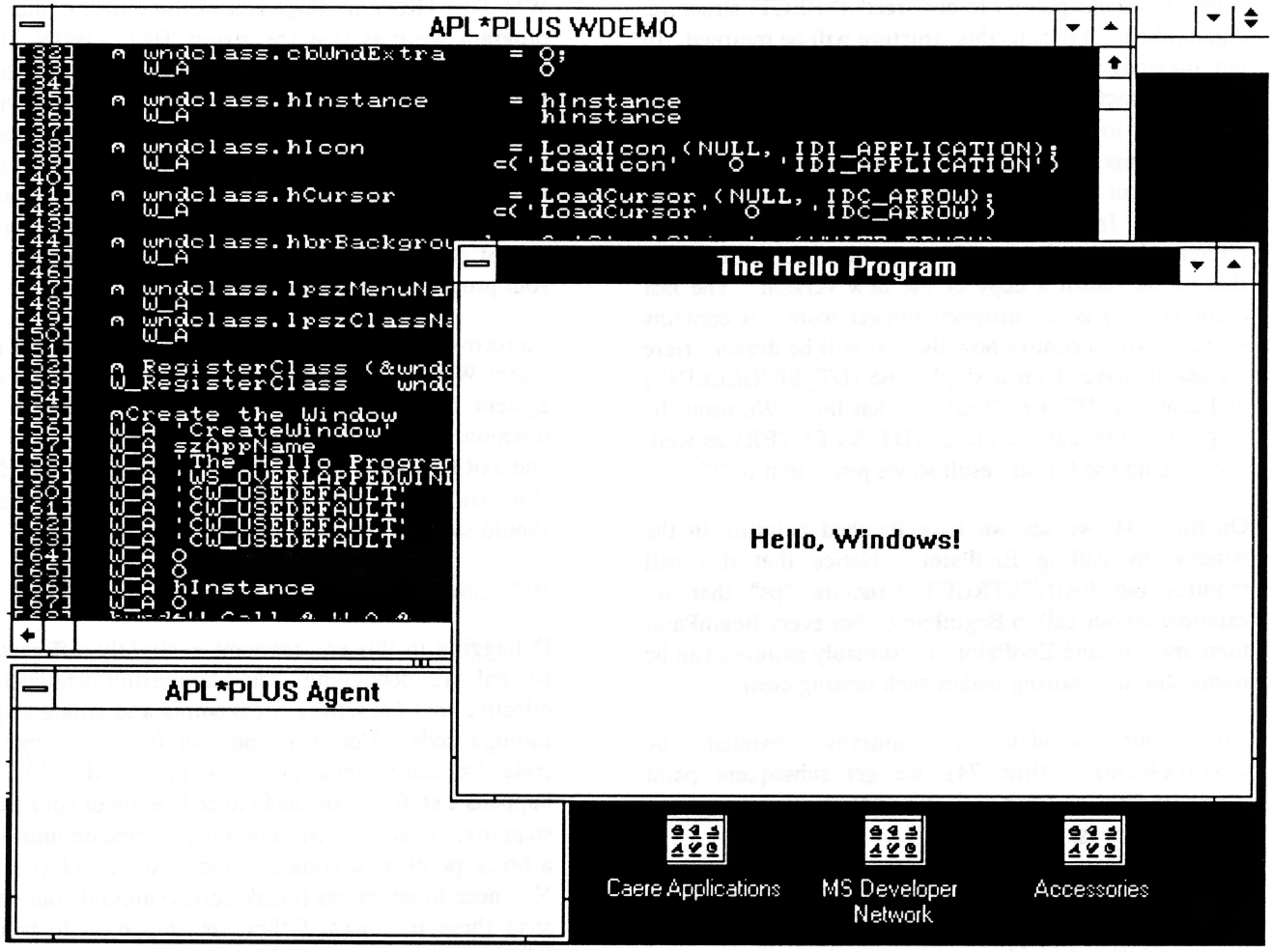
Debugging in this environment is slightly different than normal APL debugging. The Manugistics debugger is an effective tool for setting breakpoints and single stepping through code. However, you will frequently encounter code that causes messages to be processed. When this happens a shift will be made to code without your explicit stepping. That code will continue to execute until it hits a break point or it comes to the next line of your step. You need to set snares (break points) around your code to snag these instances if they are important to you. In Windows debugging, the built-in trace feature that APL has had all along can be particularly helpful. Having variables display within your code is another effective way to see what is going on.

When you're debugging paint procedures, debugging can get particularly sticky. If you reach a break point while you are painting a window, the focus goes to your APL session. If you then reselect the window, Windows will immediately send a new paint message. You can quickly get quite tangled up.

Summary

I have just explained a trivial Windows program and how you implement it using Manugistics' APLWIN interface to the Windows API. All concepts and issues exposed

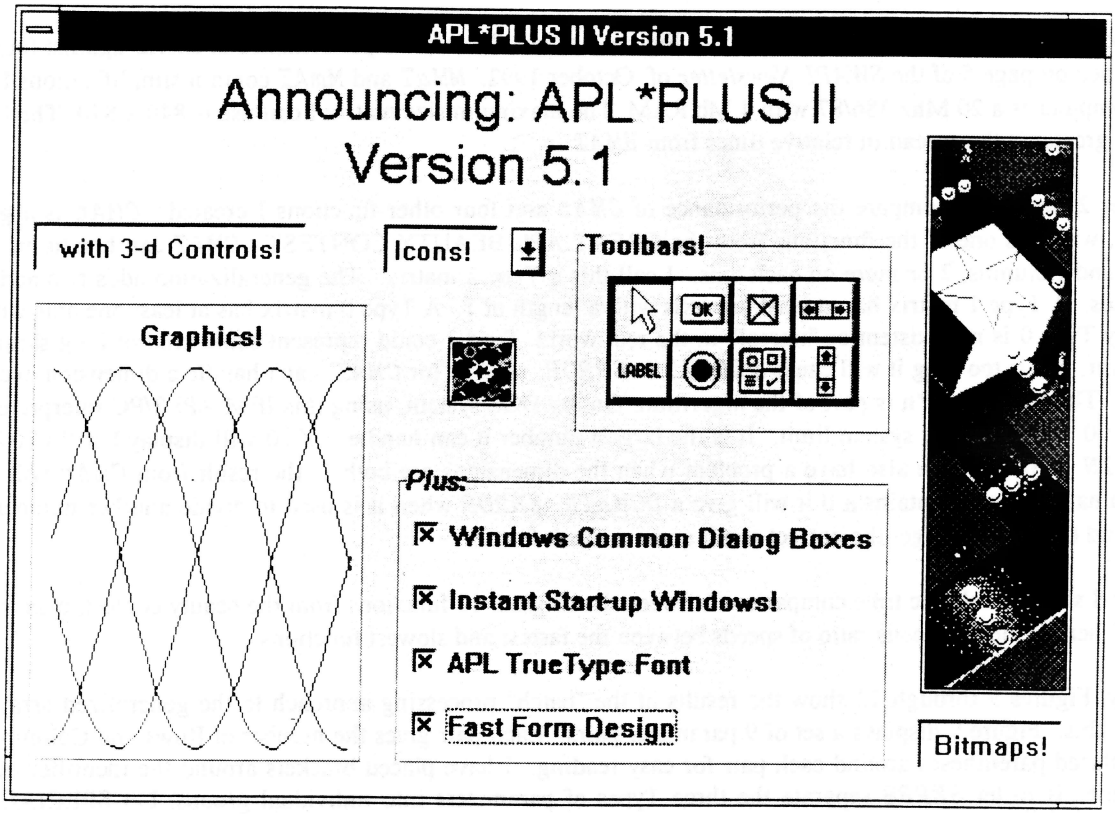
were treated in some depth. A firm understanding of the mechanics and behavior at this stage will really help when things get complicated. The APLWIN interface is a splendid way to tinker with the Windows API -- especially after you get past the stage of total disorientation. The power and flexibility of APL coupled with APLWIN give you an unbeatable combination for learning windows. Later, we'll see by using fairly simple techniques that you can optimize your code for the Windows environment and achieve acceptable (if not lightning) performance.



APL93: Taking a Closer Look

Toronto, August 15 -19

We just couldn't leave well enough alone!



APL★PLUS® II for DOS and Windows™ just got better! It's now even easier (and a lot faster!) to build applications using *all* of the features of Windows 3.1. There's even a Software Developer's Kit that shows you how to *easily* add third-party custom controls!

And don't forget that **APL★PLUS II** is *the only* APL system that provides you with a fully-integrated Debugger, Paradox Interface, full DDE access, Lotus and dBase Import/Export, optimized Assembler functions, the User Command Processor, a Numeric Data Editor and the most powerful session manager available anywhere!

Get GUIng today! Call Manugistics, Inc. at **800-592-0050, ext. 625** (in Maryland, dial 301-984-5123) or Fax 301-984-5094 and get the most out of both DOS and Windows with **APL★PLUS II Version 5.1**!

APL★PLUS is a registered trademark of Manugistics, Inc., 2115 E. Jefferson Street, Rockville MD 20852.
Other trademarks are the property of their respective owners.

PERFORMANCE ANALYSIS SUMMARY

Len Washburn
Plano, Texas

This is a continuation of the report of the results of the GREAT APL BEAUTY CONTEST which I conducted at our June 2, 1992 meeting. The first part of this report was published in the November 1992 issue of the *SWAPL Newsletter*. This part of the report deals with the objective performance of the code. The performance analysis uses 18 functions (see the listings below) in addition to the original ten listed on page 8 of the *SWAPL Newsletter* of November 1992. All code is done in IBM's APL2/PC. For those who wish to perform their own analyses, I suggest that it would be an easy job to type in the definitions of the functions.

Let's look at the results of the analysis now. Figure 1 shows the relative performance of *HMAT* against four others. *MMAT* is defined on page 5 of the *SWAPL Newsletter* of October 1992. *MMaT* and *MmAT* contain simplifications I made to *MMAT*. My computer is a 20 Mhz 386/87 with 4 Mb RAM. The maximum size matrix I can use is 840 x 840. The larger the matrix is, the greater is the spread in relative times from *HMAT* = 1.

Figures 2 through 7 compare the performance of *CMAt* and four other functions I created. *CMAt* is a generalization of *CMAT*, which is one of the functions from the GREAT APL BEAUTY CONTEST. *CMAT* can handle matrices with both Rows and Columns, 2 or more on each axis. I call this a Type 2 matrix. The generalization adds two additional Types of matrices. A Type 1 matrix has at least one axis with a length of 1. A Type 0 matrix has at least one axis with a length of 0. Clearly Type 0 is nonexistent or "unreal" in the real world. Type 1 could represent some kind of long skinny "real" object. However, if it is too long it will cause a *DOMAIN ERROR*. *CMAt* (or *CMAT*) can't handle a dimension exceeding 1024 in length. The reason is 2*n is part of the algorithm. On my DOS system, using this IBM APL2/PC interpreter, `2*1024 ≡ 1+⌊/⍳0` which is the system limit. It is the largest number it can handle. `⌊/⍳0` will display but `2*1024` will cause a *DOMAIN ERROR*. We also have a problem when the dimensions are both 1: the result from *CMAt* will be 0. When the denominator matrix contains a 0 it will give a *DOMAIN ERROR* when it is used to divide another number. The functions *alt* and *Cmat* always get the correct result and are much faster.

Figure 8 shows a relative time comparison between the original 10 functions from the beauty contest, plus *MMAT*. The thing to note here is the enormous ratio of speeds between the fastest and slowest functions.

Finally, Figures 9 through 12 show the results of the "batch" processing approach to the generalized array problem. It is simply this. Figure 9 displays a set of 9 parameter pairs. Each pair gives the number of Rows and Columns in a matrix. I have placed parentheses around each pair for easy reading. I have placed brackets around the identifier of each pair. The basic idea is to let *SEPR8* separate the three Types of parameters into individual groups. Let SPECIALIZED functions process Type 2 and Type 1 separately. Type 0 are null. So don't process them.

Let me give you a puzzle and a hint before I sign off. What is the meaning of the following sequence? `RATIO¨⎕←2/¨2*⍳10`. Suppose you are the proprietor of a picture framing shop and a customer brings in a picture requiring adjustment to the frame. Adjust the frame only, and get paid. Don't fool with the picture.

```
[0]  MMAT S;Z
[1]  Z←↑∘.+/1⌽¨1+2<⍳¨S

[0]  Z←RATIO S;C;R
[1]  Z←R×C÷2×+/(R C)←2⍴S

[0]  R MMaT C;Z
[1]  Z←(1,((R-2)⍴2),1)∘.+1,((C-2)⍴2),1

[0]  MmAT S;Z
[1]  Z←↑∘.+/1,¨((S-2)⍴¨2),¨1
```

```
[0]  Z←D ROUND A
[1]  Z←(10*-D)×⌊.5+A×10*D

[0]  alt S;Z
[1]  →(1<×/S)↓L1 ◇ Z←Sρ1,((¯2+(~S∈0 1)/S)ρ2),1 ◇ →0
[2] L1:Z←Sρ1

[0]  Z←Z_alt S;Z
[1]  →(1<×/S)↓L1 ◇ Z←Sρ1,((¯2+(~S∈0 1)/S)ρ2),1 ◇ →0
[2] L1:Z←Sρ1

[0]  Z←D part CV
[1]  ⍎(0=⎕NC 'D')/'D←''@'''
[2]  Z←(CV≠D)⊂CV

[0]  MMAT S;Z;⎕IO
[1]  Z←+∘.+/1⌽⍤1 2<⍳⍤S×⎕IO←1

[0]  NMAT A;Z
[1]  Z←+∘.+/(1+A≠1)-1⌽⍤A+⍤⊂1 1

[0]  NmAT A;Z
[1]  Z←+∘.+/1⌽⍤(A≠1)+(1+⎕IO)<⍳⍤A

[0]  CMAt S;R;⎕IO
[1]  R←(2×R-+/S∈1)-⊃+/((⍳R)⌽⍤⊂⍳R)⍉⍤(Sρ⍤2)⍉⍤1+2*(1↓⍤(⍳R←ρS)⌽⍤⊂S)ρ⍤S-1+⎕IO←0

[0]  CMat S;R;⎕IO
[1]  ⎕IO←0
[2]  'R←alt S' ⎕EA 'R←4-⊃+/((⍳R)⌽⍤⊂⍳R)⍉⍤(Sρ⍤2)⍉⍤1+2*(1↓⍤(⍳R←ρS)⌽⍤⊂S)ρ⍤S-1'

[0]  CmaT S;R;⎕IO
[1]  →(∧/S>1)/L1 ◇ alt S ◇ →0
[2] L1:R←4-⊃+/((⍳R)⌽⍤⊂⍳R)⍉⍤(Sρ⍤2)⍉⍤1+2*(1↓⍤(⍳R←ρS)⌽⍤⊂S)ρ⍤S-1+⎕IO←0

[0]  Cmat S;R;⎕IO
[1]  ⎕IO←0 ◇ →(L1,L2,L3)[+/S>1]
[2] L1:R←Sρ1 ◇ →0
[3] L2:R←Sρ1,((¯2+(~S∈0 1)/S)ρ2),1 ◇ →0
[4] L3:R←4-⊃+/((⍳R)⌽⍤⊂⍳R)⍉⍤(Sρ⍤2)⍉⍤1+2*(1↓⍤(⍳R←ρS)⌽⍤⊂S)ρ⍤S-1

[0]  Z←Z_HMAT S;C;R;Z;⎕IO
[1]  (R C)←S ◇ ⎕IO←1 ◇ Z←(R,C)ρ4 ◇ Z[;1,C]←3 ◇ Z[1,R;]←(2,C)ρ2,((C-2)ρ3),2

[0]  Z←s_r REPORT FNS;I;O;T;c;r;reps;s;⎕IO;Δ
[1]  ⍝ SYNTAX: {'P'} (Rows Cols) (reps1 reps2 ... ) REPORT 'FN1' 'FN2' ...
[2]  ⎕IO←1 ◇ O←↑↑s_r←¯3↑s_r ◇ (r c)←↑(s reps)←1↓s_r ◇ I←⍋T←reps TIMER⍤FNS
[3]  FNS←FNS~⍤⊂' crs' ⍝   {X} means parameter X is OPTIONAL.
[4]  Δ←⊂2 1ρ(' ABSOLUTE timings/RUN of:')((FNS←FNS[I]),[0.5]T←T[I])
[5]  Z←Δ,⊂2 1ρ(' RELATIVE timings/RUN of:')(FNS,[0.5]2 ROUND T÷⌊/T)
[6]  0ρZ←(∨/' '≠Z)⌿Z←⍕2 1ρ(⊂'   R = ',(⍕r),'   C = ',⍕c),⍎(0∈'Pp')↓'⊂Z'
```

```
[0] SCRIPT;F01;F02;F03;F04;F05;F06;F07;F08;F09;F10;F11;F12;F13;F14;II;PP;T1;T2;Δ
[1]  PARS←(4 5)(1 1)(1 2)(0 0)(1 0)(2 2)(0 1)(2 1)(5 13) ⋄ Δ←'@NMAT s@NmAT s'
[2]  F01←(2ρ840)10 REPORT FL1←part 'r GMAT c@r HMAT c@MMAT s@MmAT s@r MMaT c',Δ
[3]  F02←(0 0)100 REPORT FL2←'#' part 'alt s#Cmat s#CmaT s#CMAt s#CMat s'
[4]  F03←(1 1)100 REPORT FL2 ⋄ F04←(1 2)100 REPORT FL2
[5]  F05←(1 20)100 REPORT FL2 ⋄ F06←(1 200)100 REPORT FL2
[6]  F07←(1 1024)(100 100 100 1 1)REPORT FL2 ⋄ FL3←'r AMAT c&r BMAT c&CMAT s'
[7]  FL3←FL3,'&r DMAT c&EMAT s&FMAT s&r GMAT c&r HMAT c&IMAT s&JMAT s&MMAT s'
[8]  F08←'P'(2ρ340)(10 10 1 10 1,6ρ10)REPORT FL3←'&' part FL3
[9]  F09←PARS,[0.5-~?1]⍳ρPARS ⋄ F10←⊃¨(Type2 Type1 Type0)←SEPR8 PARS
[10] F11←II,¨⊂¨Z_HMAT¨↑(PP II)←Type2 ⋄ F12←II,¨⊂¨Z_alt¨↑(PP II)←Type1
[11] RESULTS←F01 F02 F03 F04 F05 F06 F07 F08 F09 F10 F11 F12

[0]  Z←SEPR8 pars;C;J;K;P
[1]  P←(pars)(⍳ρpars) ⋄ K←~C←1∈¨↑Z←(~J←0∈¨pars)/¨P ⋄ Z←(K/¨Z)(C/¨Z)(J/¨P)

[0]  Tavg←reps TIMER expression;T;WORK;⎕IO
[1]  ⎕IO←1 ⋄ expression←reps 1ρ⊂expression
[2]  ⍞←' ',⎕FX⍉,['']('WORK')(expression)
[3]  T←⎕AI[2] ⋄ WORK ⋄ T←⎕AI[2]-T ⋄ Tavg←T÷reps
```

FIGURES

Figure 1: R = 840 C = 840

ABSOLUTE timings/RUN of:							*RELATIVE timings/RUN of:*						
HMAT	*GMAT*	*MMaT*	*MmAT*	*NMAT*	*MMAT*	*NmAT*	*HMAT*	*GMAT*	*MMaT*	*MmAT*	*NMAT*	*MMAT*	*NmAT*
214	269	1378	1384	1389	1407	1407	1	1.26	6.44	6.47	6.49	6.57	6.57

Figure 2: R = 0 C = 0

ABSOLUTE timings/RUN of:					*RELATIVE timings/RUN of:*				
alt	*Cmat*	*CmaT*	*CMAt*	*CMat*	*alt*	*Cmat*	*CmaT*	*CMAt*	*CMat*
4.4	5.5	6.6	31.3	33	1	1.25	1.5	7.11	7.5

Figure 3: R = 1 C = 1

ABSOLUTE timings/RUN of:					*RELATIVE timings/RUN of:*				
alt	*Cmat*	*CmaT*	*CMAt*	*CMat*	*alt*	*Cmat*	*CmaT*	*CMAt*	*CMat*
3.8	5.5	6.6	32.9	35.1	1	1.45	1.74	8.66	9.24

Figure 4: R = 1 C = 2

ABSOLUTE timings/RUN of:					*RELATIVE timings/RUN of:*				
alt	*Cmat*	*CmaT*	*CMat*	*CMAt*	*alt*	*Cmat*	*CmaT*	*CMat*	*CMAt*
7.6	9.3	10.4	35.7	38.4	1	1.22	1.37	4.7	5.05

Figure 5: R = 1 C = 20

ABSOLUTE timings/RUN of:					*RELATIVE timings/RUN of:*				
alt	*Cmat*	*CmaT*	*CMAt*	*CMat*	*alt*	*Cmat*	*CmaT*	*CMAt*	*CMat*
7.7	9.3	10.4	46.1	47.7	1	1.21	1.35	5.99	6.19

Figure 6: R = 1 C = 200

ABSOLUTE timings/RUN of:					*RELATIVE timings/RUN of:*				
Cmat	*CmaT*	*alt*	*CMAt*	*CMat*	*Cmat*	*CmaT*	*alt*	*CMAt*	*CMat*
9.9	11	12.1	156	159.9	1	1.11	1.22	15.76	16.15

```
Figure 7:   R = 1   C = 1024
ABSOLUTE timings/RUN of:     RELATIVE timings/RUN of:
alt Cmat CmaT CMAt CMat      alt Cmat CmaT CMAt  CMat
9.4 11.6 12.1  660  660        1 1.23 1.29 70.21 70.21
```

```
Figure 8:   R = 340   C = 340
ABSOLUTE timings/RUN of:
HMAT GMAT MMAT DMAT IMAT JMAT AMAT BMAT FMAT CMAT  EMAT
  54   77  253  286  351  934  939 1005 5503 52790 56130
RELATIVE timings/RUN of:
HMAT GMAT MMAT DMAT IMAT JMAT AMAT  BMAT  FMAT   CMAT   EMAT
   1 1.43 4.69  5.3  6.5 17.3 17.39 18.61 101.91 977.59 1039.44
```

```
Figure 9   (4 5) (1 1) (1 2) (0 0) (1 0) (2 2) (0 1) (2 1) (5 13)
             [1]   [2]   [3]   [4]   [5]   [6]   [7]   [8]    [9]
```

```
Figure 10  (4 5) (2 2) (5 13)   (1 1) (1 2) (2 1)   (0 0) (1 0) (0 1)
             [1]   [6]    [9]     [2]   [3]   [8]     [4]   [5]   [7]
                 { Type 2 }           { Type 1 }          { Type 0 }
```

```
Figure 11   [1]  2 3 3 3 2   [6]  2 2   [9]  2 3 3 3 3 3 3 3 3 3 3 2
Note:            3 4 4 4 3        2 2        3 4 4 4 4 4 4 4 4 4 4 3
Type 2           3 4 4 4 3                   3 4 4 4 4 4 4 4 4 4 4 3
array            2 3 3 3 2                   3 4 4 4 4 4 4 4 4 4 4 3
results                                      2 3 3 3 3 3 3 3 3 3 3 2
```

```
Figure 12    [2]  1  [3]  1 1  [8]  1
                                  1    NOTE: Type 1 Array results
```

MORE ON BOOLEAN FLAGS

Claude Henriod
Savièse, Switzerland

Gregg Taylor's article in the June 1993 issue of this newsletter reminded me of some code I had written a few years ago for embedding short bit strings, in character format, into a file on an IBM 5110 disk. The goal was not to reduce workspace storage, but to shorten the record length without using APL2 nested arrays. For example: `⍒ (⍋ CB) ΔSWX ΔSWP 14 27`. This note is mainly a response to Gregg's comment in the second paragraph on page 13, about knowing whether a specific bit is on or off. After the code listings here, there are some examples of how the code might be used.

```
     ∇ R←S ΔSW0 P;D
[1]    ⍝ SWITCHs TURNED OFF
[2]    R←S-P⊥D⌿(D←(⍴P)⍴1 0)≠P⊤S←,S
     ∇ 1991-12-01 15.07.09

     ∇ R←S ΔSW1 P;D
[1]    ⍝ SWITCHs TURNED ON
[2]    R←S+P⊥D⌿~(D←(⍴P)⍴1 0)≠P⊤S←,S
     ∇ 1991-12-01 15.07.09
```

```
     ∇ B←ΔSWB S
[1]    ⍝ STATUS CONTROL
[2]     B←⍉(0,(32⍴2))⊤,S
[3]     B←2 0⍕B,[⎕IO]0,1-⎕IO-⍳32
     ∇ 1991-12-01 15.07.09

     ∇ R←ΔSWP P
[1]    ⍝ PREPARE THE SWITCH PARAM
[2]     R←,⍉2,[⎕IO-0.5]2*(1↓P,33)-P+1
[3]     R[⎕IO]←R[⎕IO]××1↑P
     ∇ 1991-12-01 15.07.09

     ∇ R←S ΔSWT P
[1]    ⍝ TEST SWITCHs
[2]     R←⍉((⍴P)⍴1 0)≠P⊤,S
     ∇ 1991-12-01 15.07.09

     ∇ R←S ΔSWX P;B;D
[1]    ⍝ EXCHANGE SWITCHs VALUE
[2]     R←S--≠(2,⍴S)⍴P⊥D↘B,~B←(D←(⍴P)⍴1 0)≠P⊤S←,S
     ∇ 1991-12-01 15.07.09

      ⍝ The INTEGER SWITCH can be up to 32 bits
      ⎕IO ← 1
      VB ← 21648 1287421      ⍝ 2-integer switch
      ⍝ Binary verification (Last row is switch position)

      ΔSWB ⎕← VB
21648 1287421
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 1 0 0 1 0 0 1 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 1 1 0 1 0 0 1 0 0 1 1 1 1 1 1 0 1
 0 1 2 3 4 5 6 7 8 9101112131415161718192021222324252627282930313 2

      ⍝ To test switch at position 26
      VB ΔSWT ΔSWP 26
0
1

      ⍝ To test switches at positions 12 18 19 28
      VB ΔSWT ⎕← ΔSWP 12 18 19 28
2 32 2 1 2 256 2 16
0 1 0 1
1 0 1 1

      ⍝ To force switches at positions 20 and 27
      ΔSWB ⎕← VB ΔSW1 ΔSWP 20 27              ⍝ Force 'ON'
21680 1291517
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 1 0 0 1 0 1 1 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 1 1 0 1 1 0 1 0 0 1 1 1 1 1 1 0 1
 0 1 2 3 4 5 6 7 8 9101112131415161718192021222324252627282930313 2
```

```
      ΔSWB ⎕← VB ΔSW0 ΔSWP 20 27                ⍝ Force 'OFF'
17552 1287389
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 1 0 0 1 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 1 1 0 1 0 0 1 0 0 1 1 0 1 1 1 0 1
 0 1 2 3 4 5 6 7 8 9101112131415161718192021222324252627282930313 2

      ΔSWB ⎕← VB ΔSWX ΔSWP 20 27                ⍝ Force exchange
17584 1291485
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 1 0 1 1 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 1 1 0 1 1 0 1 0 0 1 1 0 1 1 1 0 1
 0 1 2 3 4 5 6 7 8 9101112131415161718192021222324252627282930313 2

      ⍝ These functions are a subset of G.P.F., copyright 1977-1993 by
      ⍝ CHI, Savièse, Switzerland
      ⍝ Initial APL code done on IBM 5110 in 1978.
      ⍝ Printed using IBM APL2/PC V1.02 on a IBM PS/2
```

⍝ *EXTENSION NOTES*

by Gregg Taylor

TOPIC: What's in a scalar?

Ah, to go back to the simple days of flat APL -- those were the days when a scalar was a scalar, and there were no problems like we have with nested APLs. Bah! Scalars are still scalars (albeit with a significantly extended data-holding capability); we have just extended our flexibility, and have to make up for it with a little more precise thinking.

In the array terminology of APL, scalars are the "points" in APL's n-dimensional space -- those curious entities that have null shape and zero dimensionality. This may be inferred in the following way:

```
      ⎕←SHAPE←⍴ 3                       ⍝ Shape of a scalar
                                        ⍝ Looks like nothing
      ⎕←RANK←⍴⍴SHAPE                    ⍝ Dimensionality of a scalar
0                                       ⍝ Zero-dimensional
      ⍝ Newfangled methods follow:
      SHAPE ≡ ⍳0                        ⍝ This also works
1
      ≡ 3                               ⍝ Or this (with simple scalars)
0
```

With the coming of nested arrays, the definition of a scalar did *not* change. Aside from being able to use the new *Depth* and *Match* functions (≡), you still test for a scalar in the same way, and it still reports its results accurately. What has changed with the new APLs is the ways that data elements may be combined. It is now possible to group arbitrary sets of data into objects that have the properties of scalars. These properties, most notably "scalar extension", turn out to be very handy in many cases.

Scalars used to be created in a number of ways in flat APL. Here are a few examples:

```
SCALAR ← ''⍴ ANYARRAY
SCALAR ← VECTOR[1]
SCALAR ← +/,NUMARRAY
SCALAR ← VEC1 ∧.= VEC2          ⍝ When two vectors are same length
SCALAR ← 10 ⊥ NUMVEC
```

These methods still apply, but with the new data types we are sometimes not quite prepared for the consistency between the "old" definition and the new data arrays. For instance, the following may not seem quite right (but are!):

```
      ⎕←SCALAR ← ''⍴ 'TOM' 'DICK' 'AND' 'MARY'
 TOM
      ⎕←SCALAR ← ('NOT' 'PRETZEL' 'LOGIC')[⎕IO+1]
 PRETZEL
      ⎕←SCALAR ← +/(1 2) (3 4) (5 6)
 9 12                                  ⍝ 1 2 + 3 4 + 5 6
      ⎕←SCALAR ← 'ABC' ∧.= 'ABC' 'BCA' 'CAB'
 1 0 0                                 ⍝ ('A'='ABC')∧('B'='BCA')∧('C'='CAB')
                                       ⍝ 1 0 0 ∧ 1 0 0 ∧ 1 0 0
      ⎕←SCALAR ← 10 ⊥ (2 4 6) (1 3 5)
DOMAIN ERROR                           ⍝ Not capable of handling this
```

These may be unsettling enough, but consistency also brings us:

```
      ⎕←SCALAR← ,/4 5 4 5↑¨'TOM' 'DICK' 'AND' 'MARY'
 TOM DICK AND MARY
      ⎕←SCALAR← (⊂'BCA')∈'ABC' 'BCA' 'CAB'
1                                      ⍝('BCA'≡'ABC')∨('BCA'≡'BCA')∨('BCA'≡'CAB')
                                       ⍝ 0 ∨ 1 ∨ 0
      ⎕←SCALAR←  'TOM' 'DICK' 'AND' 'MARY' ⍳ ⊂'MARY'
4                                      ⍝ ⎕IO←1 (Similar to above, ∈)
      ⎕←VECTOR←  'TOM' 'DICK' 'AND' 'MARY' ⍳ 'MARY'
5 5 5 5                                ⍝ Compare above, ⍳
```

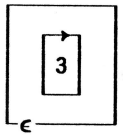
Not only is there abundant evidence that the rules governing scalars haven't changed, but I have never seen even one counter-example (and I hope I never will). However, in addition to these tried and true ways of generating scalars, the nested systems provide yet another way -- more deliberate and to the point -- to generate scalars as arbitrary collections. I have already used it above; it is known as *enclose*, and looks like this: ⊂. Monadically, this function is single-minded -- it just creates a scalar of its right argument (unless it is a depth-0 scalar -- more on this in a bit). Symbolically:

```
      SCALAR ← ⊂ ANYARRAY
      ⍴SCALAR
                                       ⍝ Null shape
      ⍴⍴SCALAR
0                                      ⍝ Zero-Dimensional
```

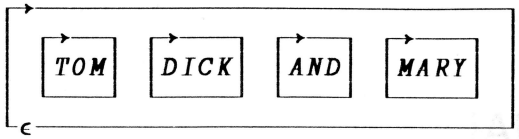
"Simple" scalars are the scalars from flat APL. These are single numbers or single characters of rank zero and depth zero. When these are enclosed, nothing happens to them. All other arrays increase their depth by one and are made scalars (if they aren't already). It may be helpful when thinking of the word "enclose" to conjure up a picture of a collection of items being put into its own container, or envelope. If it is already in an envelope, put it in another one. But, if it is a simple scalar, leave it alone. Enclosing simple scalars is a no-op. (By the way, this is one of the fundamental issues that splits APL2 enclosed arrays from Iverson's boxed arrays. In APL2, this characteristic gives rise to the *impossibility* of creating a scalar of depth one. Think about it.)

Here are a few examples of *enclose* at work:

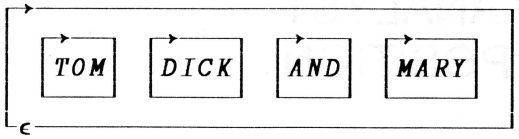
```
      DISPLAY ⊂⊂⊂⊂⊂3
3
      DISPLAY ⊂,3
```



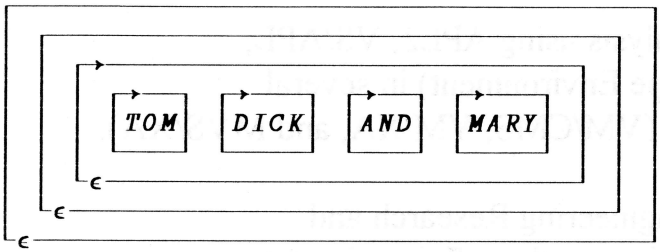
```
      DISPLAY 'TOM' 'DICK' 'AND' 'MARY'
```



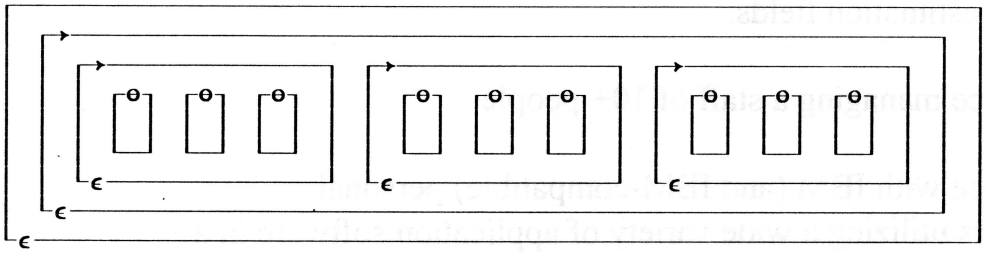

```
      DISPLAY ⊂ 'TOM' 'DICK' 'AND' 'MARY'
```




```
      DISPLAY ⊂⊂ 'TOM' 'DICK' 'AND' 'MARY'
```




```
      DISPLAY ⊂ 3⍴ ⊂ 3⍴ ⊂''
```



```
      ⍝ In the above displays, scalars are indicated by the absence of
      ⍝ an arrow (→) along the top of an enclosing box.
```

Any data array may be enclosed. The usual benefit of enclosure is to be able to use "scalar extension". Note how the enclosed item is paired with each item of the other argument for the following functions and derived functions.

```
      (⊂2 2⍴ 1 2 3 4) * 0 1 2
1 1  1 2  1  4
1 1  3 4  9 16
      (⊂¯1 ¯1)↑¨(2 2⍴⍳4)(3 3⍴⍳9)(4 4⍴⍳16)
4  9  16
      (⊂'AEIOUY')∈¨ 'TOM' 'DICK' 'AND' 'MARY'
0 0 0 1 0 0   0 0 1 0 0 0   1 0 0 0 0 0   1 0 0 0 0 1
      'TOM' 'DICK' 'AND' 'MARY' ~¨ ⊂'AEIOUY'
TM DCK ND MR
```

No, scalars aren't what they used to be. They still follow the same rules, but now they are bigger and more powerful than before. Now they can be made from (just about) any data array and still behave the way they are supposed to. It might take some getting used to, but if you pay close attention to the rules, you should be able to figure out any problem you come up against.

PROFESSIONAL APL PROGRAMMER / ANALYST SEEKS PERMANENT POSITION

Experience:

- 14 years programming and analysis using APL2, VS/APL, and APE (Application Prototype Environment) in several IBM mainframe environments (VM/CMS, VM/XA, and MVS/XA).

- 6+ years in the petroleum industry (Engineering Research and Development) writing applications in the refining simulation and cost estimation fields.

- 7 years experience managing a staff of 10+ people.

- 8 years experience with IBM (and IBM-compatible) personal computers utilizing a wide variety of application software in a Microsoft Windows environment. 6+ years experience using Banyan LAN.

- Excellent verbal and written skills.

- References available upon request.

Contact:

Merritt E. Tilley
10703 North MacArthur Boulevard, # 404
Valley Ranch (Irving), Texas 75063-5226
or call (214) 402-8792

KISS SOFTWARE INTERNATIONAL

presents **OOF**

OOF: OBJECT ORIENTED FILING
for MANUGISTICS' *APL*PLUS II/386 V5*

43 CASPER COURT
CONROE, TX. 77302
(409)273-2100

$14 95

12 OBJECT FILE FUNCTIONS
38 DERIVED OBJECT FUNCTIONS
14 APPLICATION MANAGEMENT FUNCTIONS
22 SHARE FILE FUNCTIONS
14 NATIVE FILE FUNCTIONS
11 ANCILLARY FUNCTIONS

90 PAGE REFERENCE AND TUTORIAL

OOF

OBJECT ORIENTED FILING FUNCTIONS

BASE \ DERIVED	*AUXILIARY*	*PROGRAM*	*LIBRARY*	*DATA*
OOFCI	AUXCI	PGMCI	LIBCI	DATACI
OOFCN	AUXCN	PGMCN	LIBCN	DATACN
OOFCREATE				
OOFFILE	AUXFILE	PGMFILE	LIBFILE	DATAFILE
OOFGET	AUXGET			DATAGET
OOFNAMES	AUXNAMES	PGMNAMES	LIBNAMES	DATANAMES
OOFPGMGET	AUXPGMGET	PGMGET	LIBGET	
OOFPGMGETL	AUXPGMGETL	PGMGETL	LIBGETL	
OOFPGMPUT	AUXPGMPUT	PGMPUT	LIBPUT	
OOFPUT	AUXPUT			DATAPUT
OOFRMV	AUXRMV	PGMRMV	LIBRMV	DATARMV
OOFTS	AUXTS	PGMTS	LIBTS	DATATS
		PGMRUN		

Other Functions

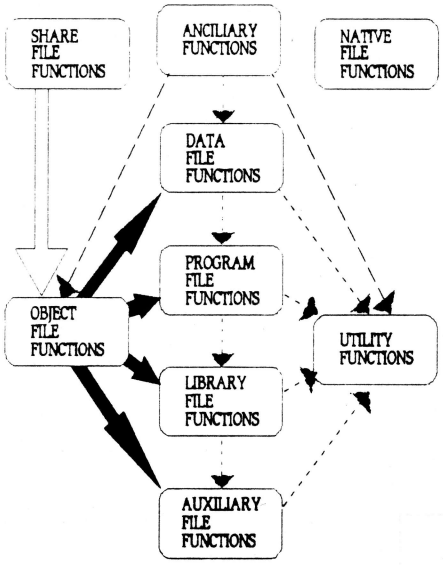
Utility Functions
FILECHGS
FILECMPR
FILELIST
FILEFNCPR
FILEGLOBALS

Program Management Functions
PGMDOC
PGMFNCPR
PGMGLOBALS
PGMLOCK
PGMSEARCH
PGMSMRY
PGMUDWS
PGMUNLOCK
PGMXREF

Ancillary Functions
CURRDIR
DAB
DLB
DTB
FL2NA
FL2PFN
KISSAKI
L2NA
NAGRADE
NASORT
RMVDUP

PRIMITIVE WRAPPER FUNCTIONS

Sharefile	*Native*
FACCESS	
FAPPEND	NAPPEND
FBDUMP	
FCLEAR	NCLEAR
FCOMPRESS	
FCREATE	NCREATE
FDROP	
FDUMP	
FDUPPROM	
FERASE	NERASE
FILES	NFILES
FINIT	NINIT
FNAMES	NNAMES
FRDCI	
FREAD	NREAD
FRENAMETO	NRENAMETO
FREPLACE	NREPLACE
FRESIZE	
FSIZE	NSIZE
FSTIE	NSWAP
FSWAP	NTIE
FUNTIE	NUNTIE



YES! Please send me _____ copies of OOF

☐ **Please bill me for $14.95 plus $3.00 postage and handling charges.(USA Only)**

☐ **Check or money order enclosed with this order form (We pay shipping on prepaid domestic orders!)**

Name: _____

Company: _____

Street: _____

City: _____ **State:** _____ **Zip:** _____

Country: _____

Phone: (___) - _____

To validate your order please sign here: _____

Mail or call: KISS Software International
43 Casper Court
Conroe, Texas 77302
Phone: (409)273-2100

☐ *3-1/2"*

☐ *5-1/4"*

I understand that if I am not fully satisfied, I may return the software within 15 days, without obligation, and KISS will cancel my bill immediately.

SouthWest APL Users' Group

Membership Application

Name: _____

Company: _____

Address: _____

_____

_____

Telephone: _____ (home) _____ (work)

E-mail: _____

Membership fees, per year: U.S.A. .....$12.
Non - U.S.A. postal supplement .....$6.
Extra Donation.....____

Please mail this application and the applicable dues to: Stuart Yarus
(Checks should be made payable to "SWAPL")
P. O. Box 210367
Bedford, Texas 76095
U. S. A.

Stuart Yarus, Editor
SWAPL Newsletter
P. O. Box 210367
Bedford, TX 76095
U. S. A.

*** *Address Correction Requested* ***

A red dot next to your name indicates that this is your LAST newsletter. Please renew your membership today!